Algoritma Strategi *Greedy*

Dr. Achmad Ridok, M.Kom dan Tim DAA

Bahasan

- **Definisi Algoritma Strategi Greedy**
- **Contoh-contoh Strategi Greedy**
  - **Contoh 1. Masalah penukaran uang**
  - **Contoh 2. Penjadwalan Sistem**
  - **Contoh 3. *Integer Knapsack***
  - **Contoh 4. Fractional Knapsack**
  - **Contoh 5. Pohon Merentang Minimum (MST)**
  - **Contoh 6. Lintasan Terpendek**
  - **Contoh 7. Huffman Code**
- **Latihan**

Metode Greedy : Definisi

- **Algoritme yang selalu mengambil solusi langsung, atau lokal, terbaik saat menemukan jawaban. Algoritme Greedy(serakah) akan selalu menemukan solusi optimal secara keseluruhan atau global untuk beberapa masalah pengoptimalan, tetapi mungkin menemukan solusi yang kurang optimal untuk beberapa contoh masalah lainnya.**

Metode Greedy : Definisi

- **Greedy = rakus, tamak, loba, …**
- **Prinsip greedy: "take what you can get now!".**
- **Algoritma greedy membentuk solusi langkah per langkah (step by step).**
- **Pada setiap langkah, terdapat banyak pilihan yang perlu dieksplorasi.**
- **Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan.**
- **Pada setiap langkah, kita membuat pilihan optimum lokal (local optimum)**
- **dengan harapan bahwa langkah sisanya mengarah ke solusi optimum global (global optimm).**

Metode Greedy : Definisi

- **Algoritma *greedy* merupakan metode yang paling populer untuk memecahkan persoalan optimasi.**

- **Persoalan optimasi (*optimization problems*):**

  ➔ persoalan mencari solusi optimum.

- **Hanya ada dua macam persoalan optimasi:**
  1. **Maksimasi (*maximization*)**
  2. **Minimasi (*minimization*)**

Metode Greedy : Definisi

- **Jika tidak ada algoritma Greedy yang selalu menemukan solusi optimal untuk suatu masalah, seseorang mungkin harus mencari (secara eksponensial) banyak solusi yang mungkin untuk menemukan solusi optimal.**
- **Algoritme greedy biasanya lebih cepat, karena tidak mempertimbangkan alternatif yang mungkin.**

Elemen-elemen algoritma greedy:

1. **Himpunan kandidat, *C*.**
2. **Himpunan solusi, *S***
3. **Fungsi seleksi (*selection function*)**
4. **Fungsi kelayakan (*feasible*)**
5. **Fungsi obyektif**

Dengan kata lain:

algoritma *greedy* melibatkan pencarian sebuah himpunan bagian, *S*, dari himpunan kandidat, *C*; yang dalam hal ini, *S* harus memenuhi beberapa kriteria yang ditentukan, yaitu menyatakan suatu solusi dan *S* dioptimisasi oleh fungsi obyektif.

Pada masalah penukaran uang:

- *Himpunan kandidat*: himpunan koin yang merepresentasikan nilai 1, 5, 10, 25, paling sedikit mengandung satu koin untuk setiap nilai.
- *Himpunan solusi*: total nilai koin yang dipilih tepat sama jumlahnya dengan nilai uang yang ditukarkan.
- *Fungsi seleksi*: pilihlah koin yang bernilai tertinggi dari himpunan kandidat yang tersisa.
- *Fungsi layak*: memeriksa apakah nilai total dari himpunan koin yang dipilih tidak melebihi jumlah uang yang harus dibayar.
- *Fungsi obyektif*: jumlah koin yang digunakan minimum.

Algoritma greedy

```
function greedy(input C: himpunan_kandidat)→ himpunan_kandidat
Deklarasi
   x : kandidat
   S : himpunan_kandidat
Algoritma:
   S ← {}    { inisialisasi S dengan kosong }
   while (not SOLUSI(S)) and (C ≠ {} ) do
        x ← SELEKSI(C) {pilih sebuah kandidat dari C}
        C ← C - {x} {elemen himpunan kandidat berkurang satu }
      if LAYAK(S ∪ {x}) then
          S ← S ∪ {x}
       endif
    endwhile {SOLUSI(S) or C = {}}
if SOLUSI(S) then return S
else output('tidak ada solusi')
endif
```

Contoh persoalan optimasi

(Masalah Penukaran Uang):
Diberikan uang senilai *A*. Tukar *A* dengan koin-koin uang yang ada. Berapa jumlah <u>minimum</u> koin yang diperlukan untuk penukaran tersebut?

➔ Persoalan minimasi

Contoh 1: tersedia banyak koin 1, 5, 10, 25

- **Uang senilai A = 32 dapat ditukar dengan banyak cara berikut:**

  32 = 1 + 1 + … + 1 (32 koin)

  32 = 5 + 5 + 5 + 5 + 10 + 1 + 1(7 koin)

  32 = 10 + 10 + 10 + 1 + 1 (5 koin)

  … dst

- **Minimum: 32 = 25 + 5 + 1 + 1 (4 koin)**

FILKOM
FAKULTAS ILMU KOMPUTER

Algoritma Strategi Greedy

Algoritma *greedy* : memecahkan masalah langkah per Langkah, pada setiap langkah:

1. **mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan (prinsip "*take what you can get now!*")**
2. **berharap bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.**

- **Contoh masalah penukaran uang:**

  Strategi *greedy*:

  Pada setiap langkah, pilihlah koin dengan nilai terbesar dari himpunan koin yang tersisa.
- **Misal: A = 32, tersedia koin: 1, 5, 10, dan 25**

  ***Langkah 1*: pilih 1 buah koin 25 (Total = 25)**

  ***Langkah 2*: pilih 1 buah koin 5**

  (Total = 25 + 5 = 30)

  ***Langkah 3*: pilih 2 buah koin 1**

  (Total = 25+5+1+1= 32)
- **Solusi: Jumlah koin minimum = 4 (solusi optimal!)**

Contoh 1. Masalah penukaran uang

- *Warning*: Optimum global belum tentu merupakan solusi optimum (terbaik), tetapi *sub-optimum* atau *pseudo-optimum*.
- Alasan:

  1. Algoritma *greedy* tidak beroperasi secara menyeluruh terhadap semua alternatif solusi yang ada (sebagaimana pada metode *exhaustive search)*.

  2. Terdapat beberapa fungsi SELEKSI yang berbeda, sehingga kita harus memilih fungsi yang tepat jika kita ingin algoritma menghasilkan solusi optiamal.
- Jadi, pada sebagian masalah algoritma *greedy* tidak selalu berhasil memberikan solusi yang optimal.

Contoh : tinjau masalah penukaran uang.

- Koin: 5, 4, 3, dan 1

  Uang yang ditukar = 7.

  Solusi *greedy*: 7 = 5 + 1 + 1 (3 koin)

  Solusi optimal: 7 = 4 + 3 (2 koin)
- Koin: 10, 7, 1

  Uang yang ditukar: 15

  Solusi *greedy*: 15 = 10 + 1 + 1 + 1 + 1 + 1 (6 koin)

  Solusi optimal: 15 = 7 + 7 + 1 (3 koin)
- Koin: 15, 10, dan 1

  Uang yang ditukar: 20

  Solusi *greedy*: 20 = 15 + 1 + 1 + 1 + 1 + 1(6 koin)

  Solusi optimal: 20 = 10 + 10 (2 koin)

Contoh 1. Masalah penukaran uang (lnjt)

- **Untuk sistem mata uang dollar AS, euro Eropa, dan *crown* Swedia, algoritma *greedy* selalu memberikan solusi optimum.**
- **Contoh: Uang $6,39 ditukar dengan uang kertas (*bill*) dan koin sen (*cent*), kita dapat memilih:**
  - **- Satu buah uang kertas senilai $5**
  - **- Satu buah uang kertas senilai $1**
  - **- Satu koin 25 sen**
  - **- Satu koin 10 sen**
  - **- Empat koin 1 sen**

$5 + $1 + 25c + 10c + 1c + 1c + 1c + 1c = $6,39

- Jika jawaban terbaik mutlak tidak diperlukan, maka algoritma *greedy* sering berguna untuk menghasilkan solusi hampiran (*approximation*), daripada menggunakan algoritma yang lebih rumit untuk menghasilkan solusi yang eksak.
- Bila algoritma *greedy* optimum, maka keoptimalannya itu dapat dibuktikan secara matematis

Contoh 1. Masalah penukaran uang (lnjt)

Masalah penukaran uang

- **Nilai uang yang ditukar: *A***
- **Himpunan koin : $\{d_1, d_2, \ldots, d_n\}$.**
- **Himpunan solusi: $X = \{x_1, x_2, \ldots, x_n\}$,**
- **$x_i = 1$ jika d_i dipilih, $x_i = 0$ jika d_i tidak dipilih.**

Obyektif persoalan adalah

Minimisasi $F = \sum_{i=1}^{n} x_i$ (fungsi obyektif)

dengan kendala $\sum_{i=1}^{n} d_i x_i = A$

Penyelesaian dengan *exhaustive search*

- Terdapat 2^n kemungkinan solusi (nilai-nilai $X = \{x_1, x_2, \ldots, x_n\}$)
- Untuk mengevaluasi fungsi obyektif = $O(n)$
- Kompleksitas algoritma *exhaustive search* seluruhnya = $O(n \cdot 2^n)$.

Contoh 1. Masalah penukaran uang (lnjt)

Strategi *greedy:*

- Pada setiap langkah, pilih koin dengan nilai terbesar dari himpunan koin yang tersisa.
- urutkan himpunan koin dalam urutan yang menurun :$O(n)$.
- tidak selalu optimal

```
function TukarCoin(input C : koinSet,A : integer)→koinSet
Deklarasi
   S : koinSet
   x : koin
Algoritma
  S ← {}
  while (∑(nilai2 semua koin di S) ≠ A) and (C ≠ {} ) do
     x ← koin dengan nilai terbesar
     C ← C - {x}
     if  (∑(nilai koin di S) + nilai koin x ≤ A then
       S ← S ∪ {x}
     endif
  endwhile
  if (∑(nilai2 koin di dalam S) = A then  return S
  else output('tidak ada solusi')
  endif
```

Contoh 2. Penjadwalan Sistem

- **Persoalan: Sebuah *server* (dapat berupa *processor*, pompa, kasir di bank, dll) mempunai *n* pelanggan (*customer*, *client*) yang harus dilayani. Waktu pelayanan untuk setiap pelanggan *i* adalah t_i.**

  Minimumkan total waktu di dalam sistem:

  $$T = \sum_{i=1}^{n} ti \quad \text{(waktu di dalam sistem)}$$

- **Ekivalen dengan meminimumkan waktu rata-rata pelanggan di dalam sistem.**

Contoh 3: Tiga pelanggan dengan

$t_1 = 5$, $t_2 = 10$, $t_3 = 3$,

Enam urutan pelayanan yang mungkin:

Urutan	*T*
1, 2, 3:	5 + (5 + 10) + (5 + 10 + 3) = 38
1, 3, 2:	5 + (5 + 3) + (5 + 3 + 10) = 31
2, 1, 3:	10 + (10 + 5) + (10 + 5 + 3) = 43
2, 3, 1:	10 + (10 + 3) + (10 + 3 + 5) = 41
3, 1, 2:	3 + (3 + 5) + (3 + 5 + 10) = 29 ← (optimal)
3, 2, 1:	3 + (3 + 10) + (3 + 10 + 5) = 34

Contoh 2. Penjadwalan Sistem

Solusi Exhaustive Search:

- **Urutan pelangan yang dilayani oleh server merupakan suatu permutasi . Jika ada *n* orang pelanggan, maka tedapat *n*!**
- **urutkan pelanggan untuk mengevaluasi fungsi obyektif : *O*(*n*)**
- **Kompleksitas algoritma *exhaustive search* = O(*nn*!)**

Solusi Greedy :

Strategi greedy: Pada setiap langkah, pilih pelanggan yang membutuhkan waktu pelayanan terkecil di antara pelanggan lain yang belum dilayani.

Contoh 2. Penjadwalan Sistem (lanjt)

- **Algoritma *greedy* untuk penjadwalan pelanggan akan selalu menghasilkan solusi optimum.**
- **Teorema. Jika $t_1 \leq t_2 \leq \ldots \leq t_n$ maka pengurutan $i_j = j,\ 1 \leq j \leq n$ meminimumkan**

$$T = \sum_{k=1}^{n} \sum_{j=1}^{k} t_{i_j}$$

untuk semua kemungkinan permutasi i_j.

```
function Penjadwalan (input C : objekSet)→ objekSet
Deklarasi
    S : objekSet
    i : Objek
Algoritma
   S ← {}
   while (C ≠ {}) do
      i ← objek dengan t[i] terkecil
      C ← C - {i}
      S ← S ∪ {i}
   endwhile
   return S
```

Contoh 3. *Integer Knapsack*

Maksimasi $F = \sum_{i=1}^{n} p_i x_i$

dengan kendala (*constraint*)

$$\sum_{i=1}^{n} w_i x_i \leq K$$

yang dalam hal ini, x_i = 0 atau 1, $i = 1, 2, \ldots, n$

Contoh 3. *Integer Knapsack (Lanjt)*

Solusi *exhaustive search:*

- **Sudah dijelaskan pada pembahasan exhaustive search.**
- **Kompleksitas algoritma *exhaustive search* untuk persoalan ini = $O(n \cdot 2^n)$.**

Solusi *greedy* :

- **Masukkan objek satu per satu ke dalam *knapsack*. Sekali objek dimasukkan ke dalam *knapsack*, objek tersebut tidak bisa dikeluarkan lagi.**
- **Terdapat beberapa strategi *greedy* yang heuristik yang dapat digunakan untuk memilih objek yang akan dimasukkan ke dalam *knapsack*:**

Contoh 3. *Integer Knapsack (Lanjt)*

1. ***Greedy by profit*.**
   - **Pada setiap langkah, pilih objek yang mempunyai keuntungan terbesar.**
   - **Mencoba memaksimumkan keuntungan dengan memilih objek yang paling menguntungkan terlebih dahulu.**
2. ***Greedy by weight.***
   - **Pada setiap langkah, pilih objek yang mempunyai berat teringan.**
   - **Mencoba memaksimumkan keuntungan dengan dengan memasukkan sebanyak mungkin objek ke dalam *knapsack*.**
3. ***Greedy by density*.**
   - **Pada setiap langkah, *knapsack* diisi dengan objek yang mempunyai p_i / w_i terbesar.**
   - **Mencoba memaksimumkan keuntungan dengan memilih objek yang mempunyai keuntungan per unit berat terbesar.**

Pemilihan objek berdasarkan salah satu dari ketiga strategi di atas tidak menjamin akan memberikan solusi optimal.

Contoh 3. *Integer Knapsack (Lanjt)*

Contoh 1

$w1 = 6$; $p1 = 12$; $w2 = 5$; $p2 = 15$;

$w3 = 10$; $p3 = 50$; $w4 = 5$; $p4 = 10$

Kapasitas *knapsack* $K = 16$

Properti objek				*Greedy by*			Solusi Optimal
i	w_i	p_i	p_i/w_i	*profit*	*weight*	*density*	
1	6	12	2	0	1	0	0
2	5	15	3	1	1	1	1
3	10	50	5	1	0	1	1
4	5	10	2	0	1	0	0
Total bobot				15	16	15	15
Total keuntungan				65	37	65	65

- Solusi optimal: $X = (0, 1, 1, 0)$
- *Greedy by profit* dan *greedy by density* memberikan solusi optimal!

Contoh 3. *Integer Knapsack (Lanjt)*

Contoh 2.

***w1* = 100; *p1* = 40;**

***w2* = 50; *p2* = 35;**

***w3* = 45; *p3* = 18;**

***w4* = 20; *p4* = 4;**

***w5* = 10; *p5* = 10;**

***w6* = 5; *p6* = 2**

Kapasitas *knapsack K* = 100

- ✓ **Ketiga strategi gagal memberikan solusi optimal!**

Properti objek				*Greedy by*			Solusi Optimal
i	w_i	p_i	p_i/w_i	*profit*	*weight*	*density*	
1	100	40	0,4	1	0	0	0
2	50	35	0,7	0	0	1	1
3	45	18	0,4	0	1	0	1
4	20	4	0,2	0	1	1	0
5	10	10	1,0	0	1	1	0
6	5	2	0,4	0	1	1	1
Total bobot				100	80	85	100
Total keuntungan				40	34	51	55

Kesimpulan: Algoritma *greedy* tidak selalu berhasil menemukan solusi optimal untuk masalah 0/1 *Knapsack*.

Contoh 4. Fractional Knapsack

Maksimasi $F = \sum_{i=1}^{n} p_i x_i$

dengan kendala (*constraint*)

$$\sum_{i=1}^{n} w_i x_i \le K$$

yang dalam hal ini, $0 \le x_i \le 1$, $i = 1, 2, \ldots, n$

Contoh 4. Fractional Knapsack (Lanjut)

Solusi *exhaustive search:*

- **Oleh karena $0 \leq x_i \leq 1$, maka terdapat tidak berhinga nilai-nilai x_i.**
- **Persoalan *Fractional Knapsack* menjadi malar (*continuous*) sehingga tidak mungkin dipecahkan dengan algoritma *exhaustive search.***

Solusi *greedy:*

- **Ketiga strategi *greedy* yang telah disebutkan di atas dapat digunakan untuk memilih objek yang akan dimasukkan ke dalam *knapsack*.**

Contoh 4. Fractional Knapsack (Lanjut)

Contoh
***w*1 = 18; *p*1 = 25;**
***w*2 = 15; *p*2 = 24**
***w*3 = 10; *p*3 = 15**
Kapasitas *knapsack K* = 20

Solusi optimal:

***X* = (0, 1, 1/2) dengan keuntungan maksimum = 31,5.**

Properti objek				*Greedy by*		
i	w_i	p_i	p_i/w_i	*profit*	*weight*	*density*
1	18	25	1,4	1	0	0
2	15	24	1,6	2/15	2/3	1
3	10	15	1,5	0	1	1/2
Total bobot				20	20	20
Total keuntungan				28,2	31,0	31,5

Contoh 4. Fractional Knapsack (Lanjut)

- **Strategi pemilihan objek berdasarkan densitas p_i/w_i terbesar akan selalu memberikan solusi optimal.**
- **Agar proses pemilihan objek berikutnya optimal, maka kita urutkan objek berdasarkan p_i/w_i yang menurun, sehingga objek berikutnya yang dipilih adalah objek sesuai dalam urutan itu.**
- **Teorema 3.2. Jika $p_1/w_1 \geq p_2/w_2 \geq \ldots \geq p_n/w_n$ maka algoritma *greedy* dengan strategi pemilihan objek berdasarkan p_i/w_i terbesar menghasilkan solusi yang optimum.**

Algoritma fractional knapsack:

1. **Hitung harga pi/wi , i = 1, 2, ..., n**
2. **Urutkan seluruh objek berdasarkan nilai pi/wi dari besar ke kecil**
3. **Panggil FractinonalKnapsack**

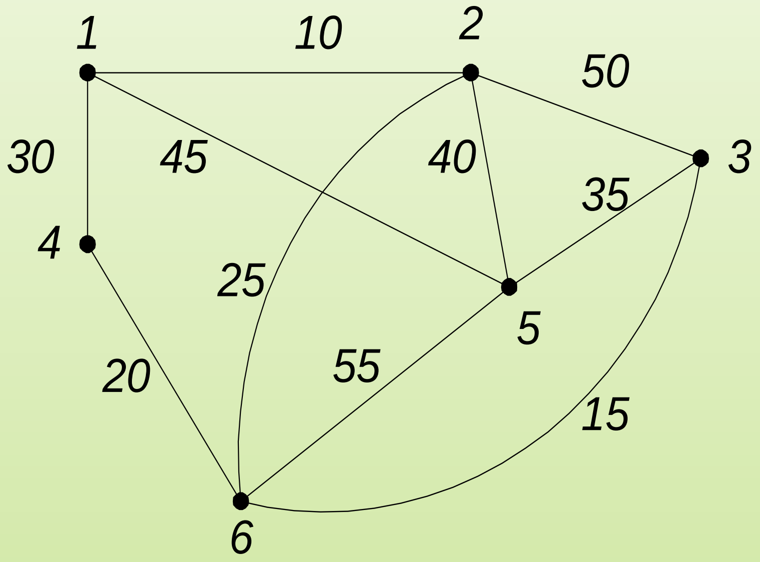
Contoh 4. Fractional Knapsack (Lanjut)

```
function FKnapsack(
     input C : objek2, K : real) →
solusi2
Deklarasi
    i, TotalBobot : integer;
    MasihMuatUtuh : boolean
    x : himpunan_solusi
Algoritma:
   for i ← 1 to n do
      x[i] ← 0 {inisialisaso}
   endfor
   i ← 0; TotalBobot ← 0;
   MasihMuatUtuh ← true
```

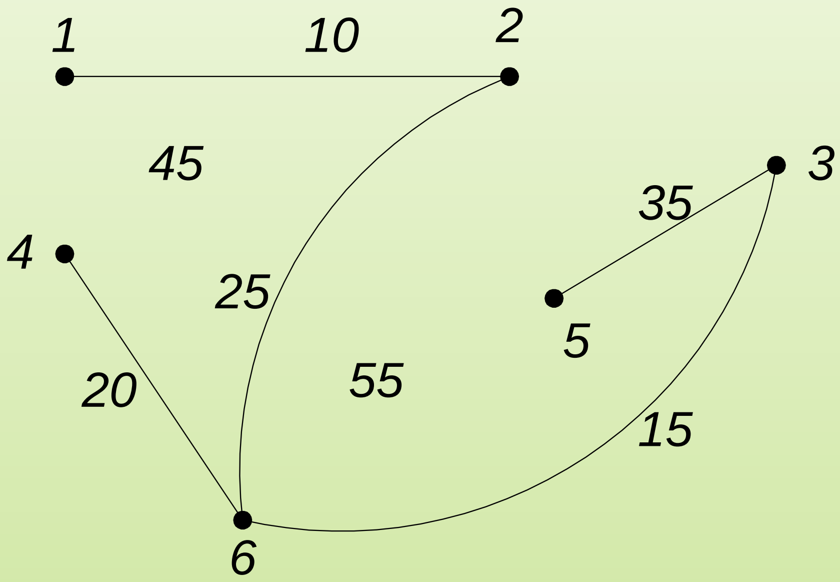
```
while (i ≤ n) and (MasihMuatUtuh) do
  i ← i + 1 { tinjau objek ke-i }
  if TotalBobot + C.w[i] ≤ K then
     {objek i masukkan ke knapsack}
     x[i] ← 1
     TotalBobot ← TotalBobot + C.w[i]
  else
    MasihMuatUtuh ← false
     x[i] ← (K – TotalBobot)/C.w[i]
  endif
endwhile { i > n or not MasihMuatUtuh }
return x
```

Kompleksitas waktu algoritma = $O(n)$.

Contoh 5. Pohon Merentang Minimum (MST)



(a) Graf *G = (V, E)*



(b) Pohon merentang minimum

FILKOM UB — FAKULTAS ILMU KOMPUTER

Contoh 5. MST (Lanjut)

***Algoritma Prim* (Strategi *greedy*) :**

Pada setiap langkah, pilih sisi *e* dari graf *G*(*V*, *E*) yang mempunyai bobot terkecil dan bersisian dengan simpul-simpul di *T* tetapi *e* tidak membentuk sirkuit di *T*.

- **Komplesiats algoritma: $O(n^2)$**

Q ← V(G)
for each $u \in Q$ do key(u) ← ∞
$key(r)$ ← 0; $\pi(r)$ ← Nil;
while Q $\neq \emptyset$ do
    u ← Extract-Min(Q)
    for each v adjacent to u do //total 2|E| times
      if $v \in Q$ and $w(u,v) < key(v)$ //membership bit
        then $\pi(v) \leftarrow u$;
        $key(v) \leftarrow w(u,v)$ //Decrease-Key

Priority queue ← binary heap

- **Extract-Min, Decrease-Key = O(log |V|)**

Total run-time = O(|E|log |V|)

Contoh 5. MST (Lanjut)

- MST (G) =
  - pohon: subgraf asiklik dari G
  - spanning: spans (menghubungkan) semua simpul dari G
  - memiliki bobot total minimum = jumlah bobot sisi
- G adalah matroid → GA memberikan pohon rentang maksimum
  - bagaimana GA dapat menemukan pohon rentang minimum?
- Seleksi Greedy edge
  - Kruskal: antara dua komponen yang terhubung
  - Prim's: from connected component to new vertex

- Strategi *greedy* yang digunakan:

  Pada setiap langkah, pilih sisi *e* dari graf *G* yang mempunyai bobot minimum tetapi *e* tidak membentuk sirkuit di *T*.

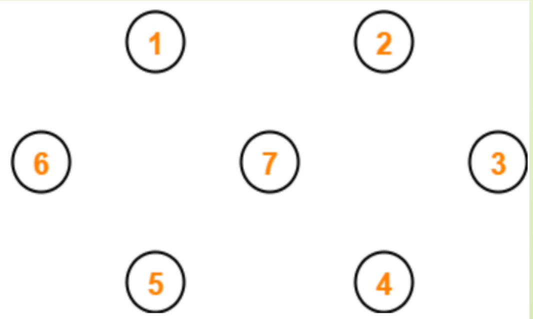
Kompleksitas algoritma: $O(|E| \log |E|)$

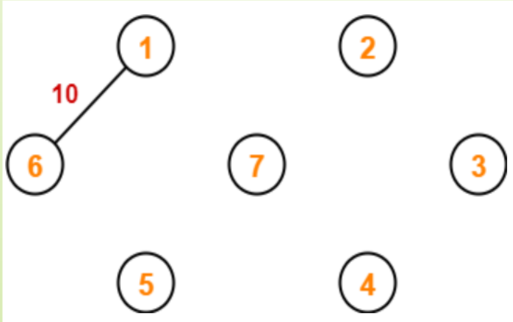
Contoh 5. MST (Lanjut)

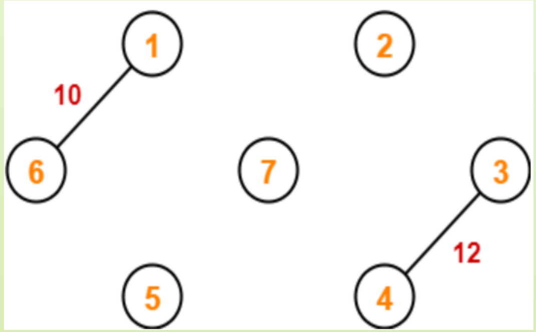
- **Algoritma Kruskal:**
- **Urutkan semua edge yang terdapat dalam graf berdasarkan bobotnya dari yang terkecil ke yang terbesar.**
- **Mulai dari edge dengan bobot terkecil, tambahkan edge tersebut ke MST jika edge tersebut tidak membentuk siklus dengan edge-edge sebelumnya yang telah dipilih. Jika membentuk siklus, edge tersebut diabaikan.**
- **Ulangi langkah 2 sampai MST terbentuk.**

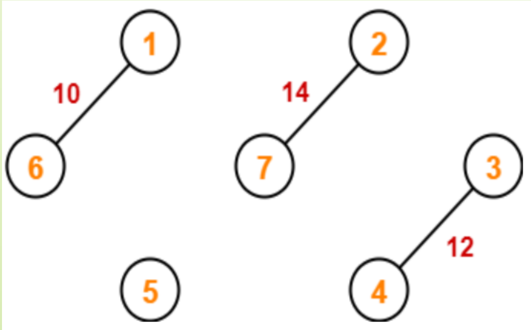
- **Algoritma Prim:**
- **Pilih node awal secara acak dan tambahkan ke MST.**
- **Cari edge dengan bobot terkecil yang menghubungkan node yang sudah ada dalam MST dengan node yang belum ada dalam MST.**
- **Tambahkan node tersebut ke MST dan ulangi langkah 2 sampai semua node dalam graf telah ditambahkan ke MST.**

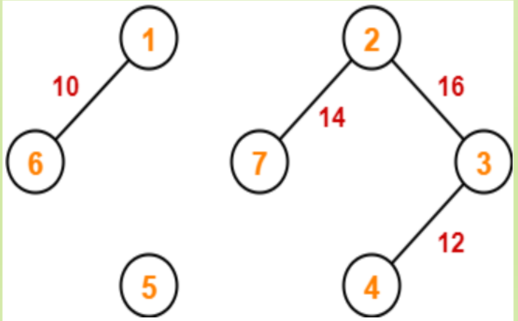
Algoritma Kruskal - MST

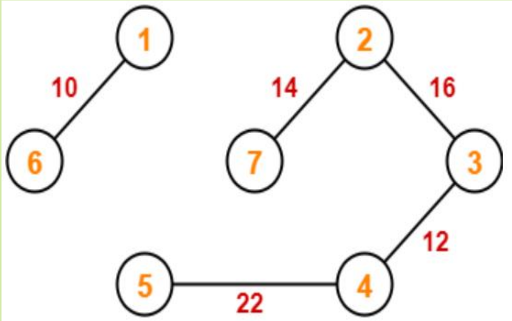


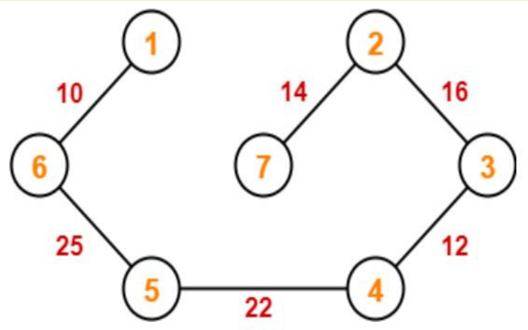


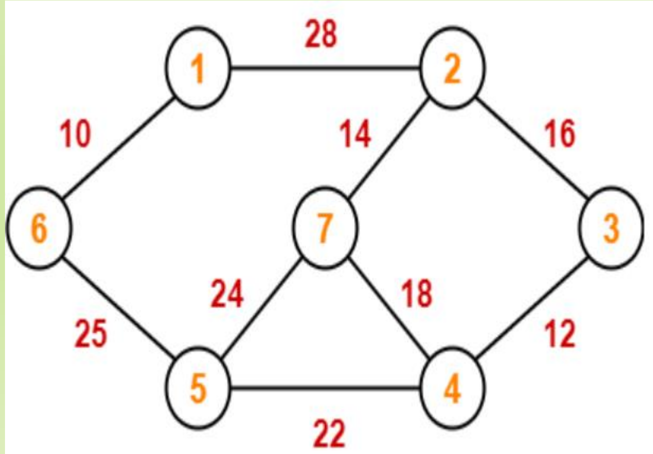




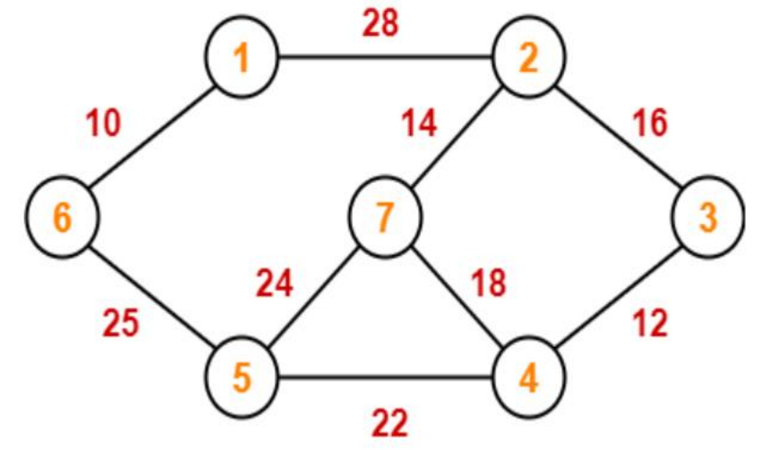


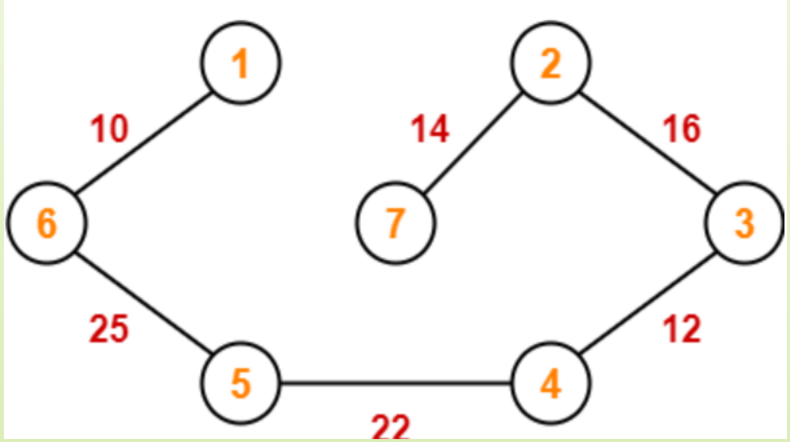


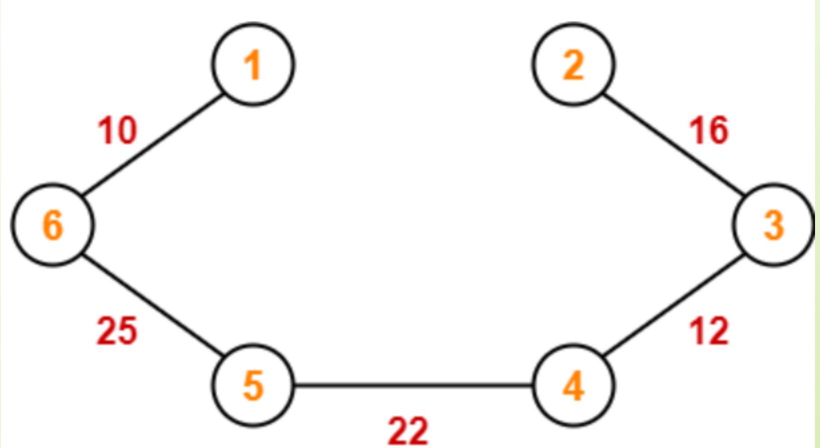


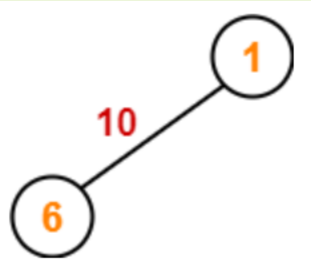


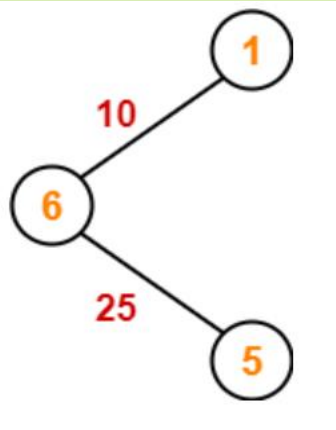
Algoritma Prim - MST

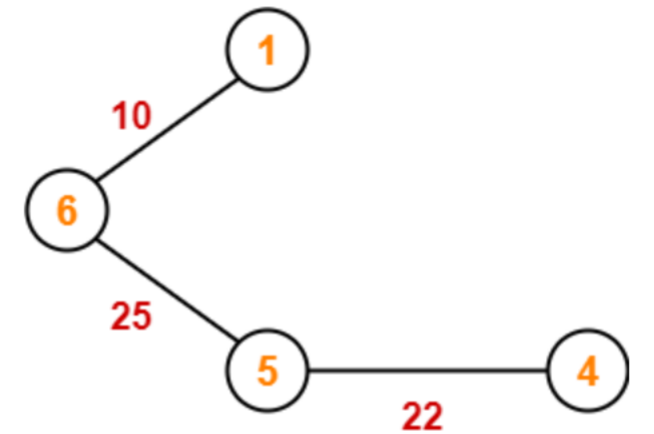


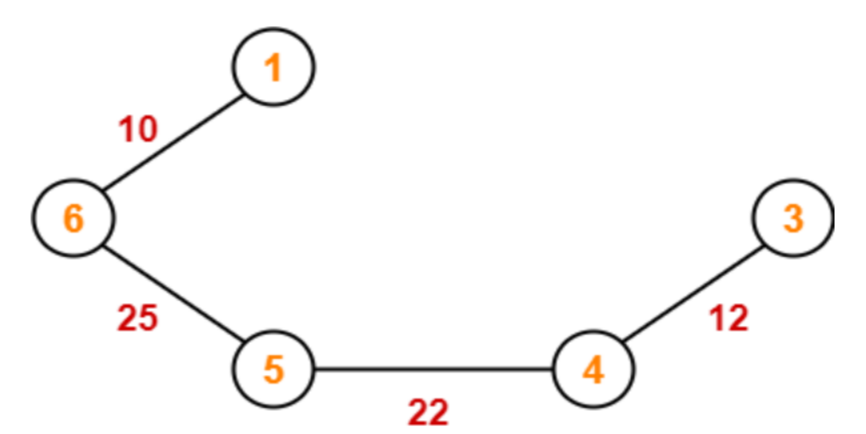












6. Lintasan Terpendek (*Shortest Path)*

Beberapa macam persoalan lintasan terpendek:

- **Lintasan terpendek antara dua buah simpul tertentu (*a pair shortest path*).**
- **Lintasan terpendek antara semua pasangan simpul (*all pairs shortest path*).**
- **Lintasan terpendek dari simpul tertentu ke semua simpul yang lain (*single-source shortest path*).**
- **Lintasan terpendek antara dua buah simpul yang melalui beberapa simpul tertentu (*intermediate shortest path*).**

Persoalan:

Diberikan graf berbobot $G = (V, E)$. Tentukan lintasan terpendek dari sebuah simpul asal *a* ke setiap simpul lainnya di *G*.

Asumsi yang kita buat adalah bahwa semua sisi berbobot positif.

6. Lintasan Terpendek (*Shortest Path*)

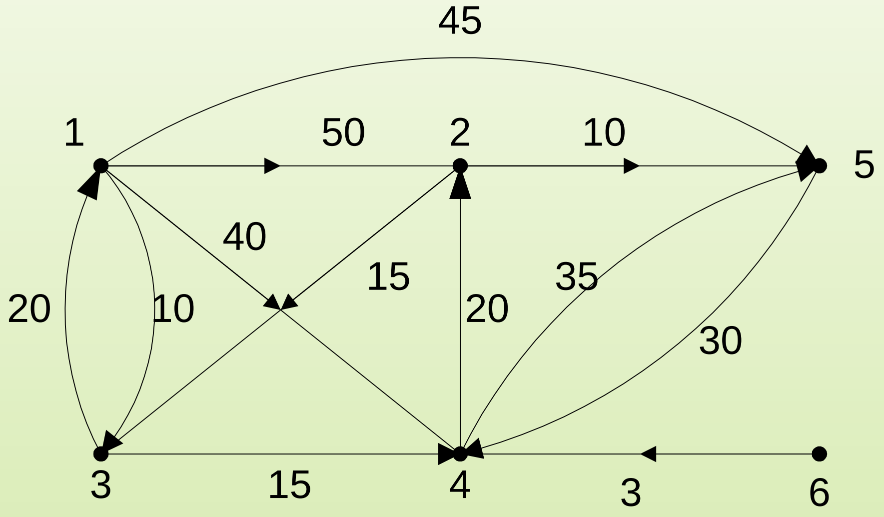
```
fungsi Dijkstra(Graf, asal):
    Q adalah himpunan titik

    untuk setiap titik v dalam Graf:
        jarak[v] ← tak hingga
        sebelum[v] ← kosong
        tambahkan v ke dalam Q
    jarak[asal] ← 0;

    selama Q tidak kosong:
        u ← titik dalam Q dengan nilai jarak[u] terkecil
        hapus u dari Q

        untuk setiap tetangga v dari u: // hanya v yang masih dalam Q
            alt ← jarak[u] + jarak_antara(u, v)
            jika alt < jarak[v]:
                jarak[v] ← alt
                sebelum[v] ← u

kembalikan jarak[], sebelum[]
```

Algoritme Dijkstra : suatu algoritma Greedy untuk menemukan jalur terpendek dengan solusi optimal.

Contoh 6. Lintasan Terpendek (Lanjutan)

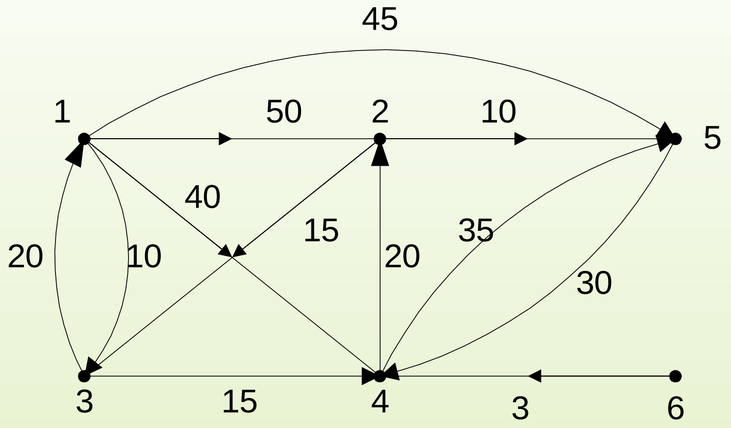
Strategi *greedy*:

Lintasan dibentuk satu per satu. Lintasan berikutnya yang dibentuk ialah lintasan yang meminimumkan jumlah jaraknya.



Simpul asal	Simpul tujuan	Lintasan terpendek	Jarak
1	3	$1 \rightarrow 3$	10
1	4	$1 \rightarrow 3 \rightarrow 4$	25
1	2	$1 \rightarrow 3 \rightarrow 4 \rightarrow 2$	45
1	5	$1 \rightarrow 5$	45
1	6	tidak ada	-

6. Lintasan Terpendek



Algoritma Dijkstra

Strategi *greedy*:

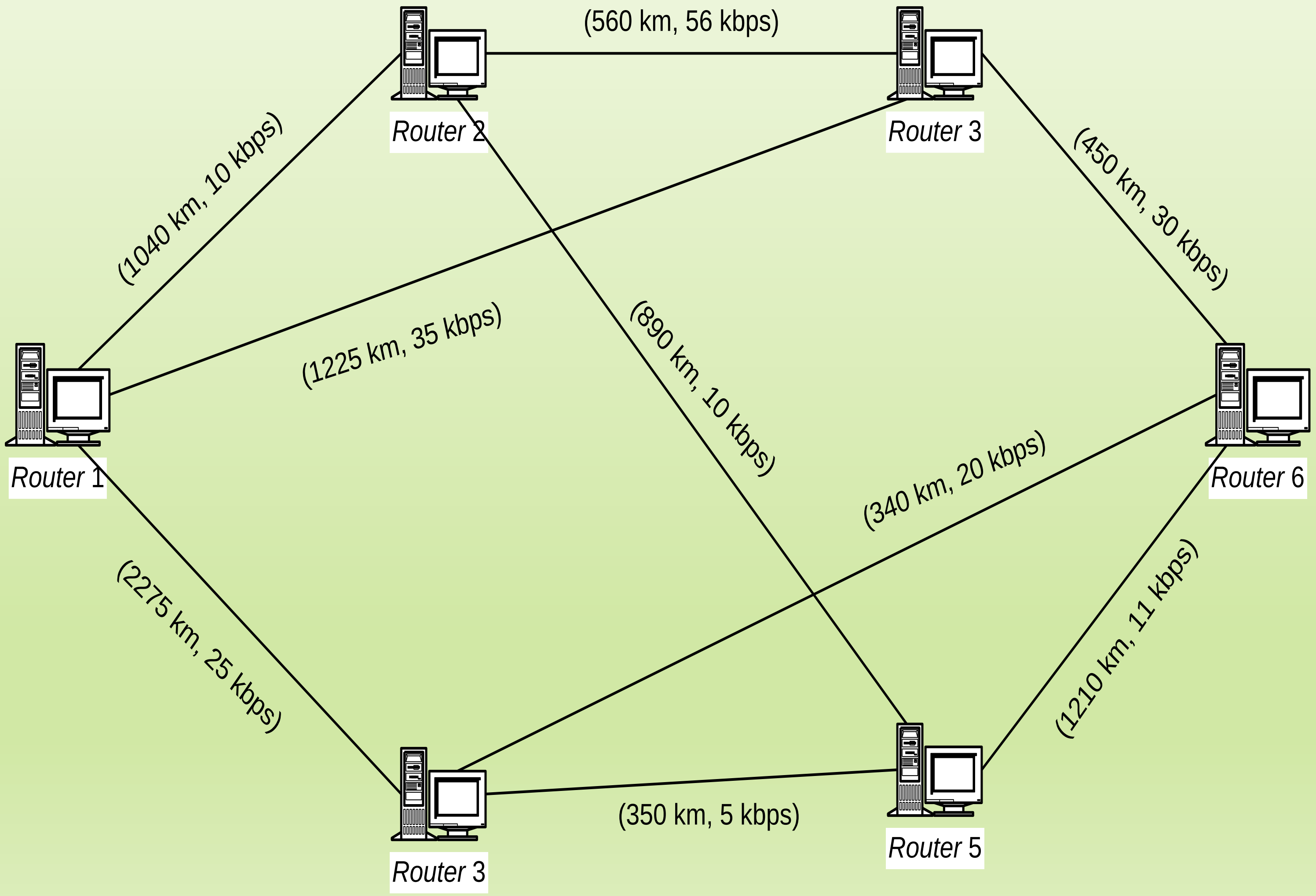
- **Pada setiap langkah, ambil sisi yang berbobot minimum yang menghubungkan sebuah simpul yang sudah terpilih dengan sebuah simpul lain yang belum terpilih.**
- **Lintasan dari simpul asal ke simpul yang baru haruslah merupakan lintasan yang terpendek diantara semua lintasannya ke simpul-simpul yang belum terpilih.**

Lelaran	Simpul yang dipilih	Lintasan	*S*						*D*					
			1	2	3	4	5	6	1	2	3	4	5	6
Inisial	-	-	0	0	0	0	0	0	0	50 *(1,2)*	10 *(1,3)*	40 *(1,4)*	45 *(1,5)*	∞ *(1,6)*
1	1	1	1	0	0	0	0	0	∞	50 *(1,2)*	10 *(1,3)*	40 *(1,4)*	45 *(1,5)*	∞ *(1,6)*
2	3	1, 3	1	0	1	0	0	0	∞	50 *(1,2)*	10 *(1,3)*	25 *(1,3,4)*	45 *(1,5)*	∞ *(1,6)*
3	4	1, 3, 4	1	0	1	1	0	0	∞	45 *(1,3,4,2)*	10 *(1,3)*	25 *(1,3,4)*	45 *(1,5)*	∞ *(1,6)*
4	2	1, 3, 4, 2	1	1	1	1	0	0	∞	45 *(1,3,4,2)*	10 *(1,3)*	25 *(1,3,4)*	45 *(1,5)*	∞ *(1,6)*
5	5	1, 5	1	1	1	1	1	0	∞	45	10	25	45	∞

Contoh 6. Lintasan Terpendek (Lanjutan)

Aplikasi algoritma Dijkstra:

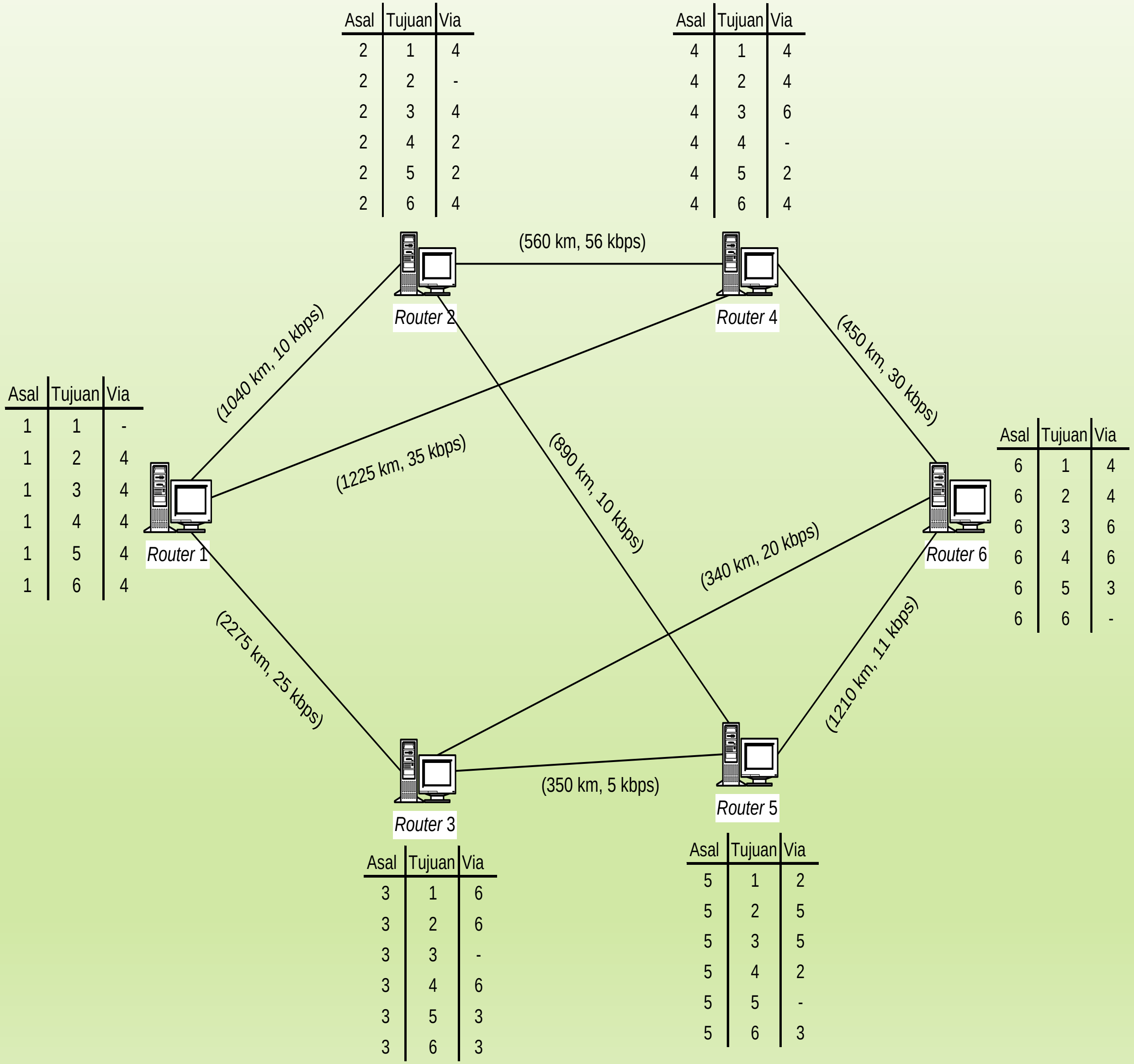
→*Routing* pada jaringan komputer



Lintasan terpendek (berdasarkan delai):

Router Asal	*Router* Tujuan	Lintasan Terpendek
1	1	-
	2	1, 4, 2
	3	1, 4, 6, 3
	4	1, 4
	5	1, 4, 2, 5
	6	1, 4, 6
2	1	2, 4, 1
	2	-
	3	2, 4, 6, 3
	4	2, 4
	5	2, 5
	6	2, 4, 6
3	1	3, 6, 4, 1
	2	3, 6, 4, 2
	3	-
	4	3, 6, 4
	5	3, 5
	6	3, 6
4	1	4, 1
	2	4, 2
	3	4, 6, 2
	4	4, 6, 3
	5	4, 2, 5
	6	4, 6

Router Asal	*Router* Tujuan	Lintasan Terpendek
5	1	5, 2, 4, 1
	2	5, 2
	3	5, 3
	4	5, 2, 4
	5	-
	6	5, 3, 6
6	1	6, 4, 1
	2	6, 4, 2
	3	6, 3
	4	6, 4
	5	6, 3, 5
	6	-



FILKOM
FAKULTAS ILMU KOMPUTER

Contoh 7. Algoritm Pemampatan (*Huffman code*)

Contoh :

Fixed-length code

Karakter	*a*	*b*	*c*	*d*	*e*	*f*
Frekuensi	45%	13%	12%	16%	9%	5%
Kode	000	001	010	011	100	111

'bad' dikodekan sebagai '001000011'

Pengkodean 100.000 karakter membutuhkan 300.000 bit.

Variable-length code (Huffman code)

Karakter	*a*	*b*	*c*	*d*	*e*	*f*
Frekuensi	45%	13%	12%	16%	9%	5%
Kode	0	101	100	111	1101	1100

'bad' dikodekan sebagai '1010111 '

Pengkodean 100.000 karakter membutuhkan
$(0{,}45 \times 1 + 0{,}13 \times 3 + 0{,}12 \times 3 + 0{,}16 \times 3 + 0{,}09 \times 4 + 0{,}05 \times 4) \times 100.000 = 224.000$ bit

Nisbah pemampatan:

$(300.000 – 224.000)/300.000 \times 100\% = 25{,}3\%$

Contoh 7. Huffman Code (Lanjut)

Prinsip kode Huffman:

- **karakter yang paling sering muncul beri kode yang lebih pendek;**
- **karakter yang relatif jarang muncul beri kode yang lebih panjang.**

Algoritma Greedy untuk Membentuk Kode Huffman:

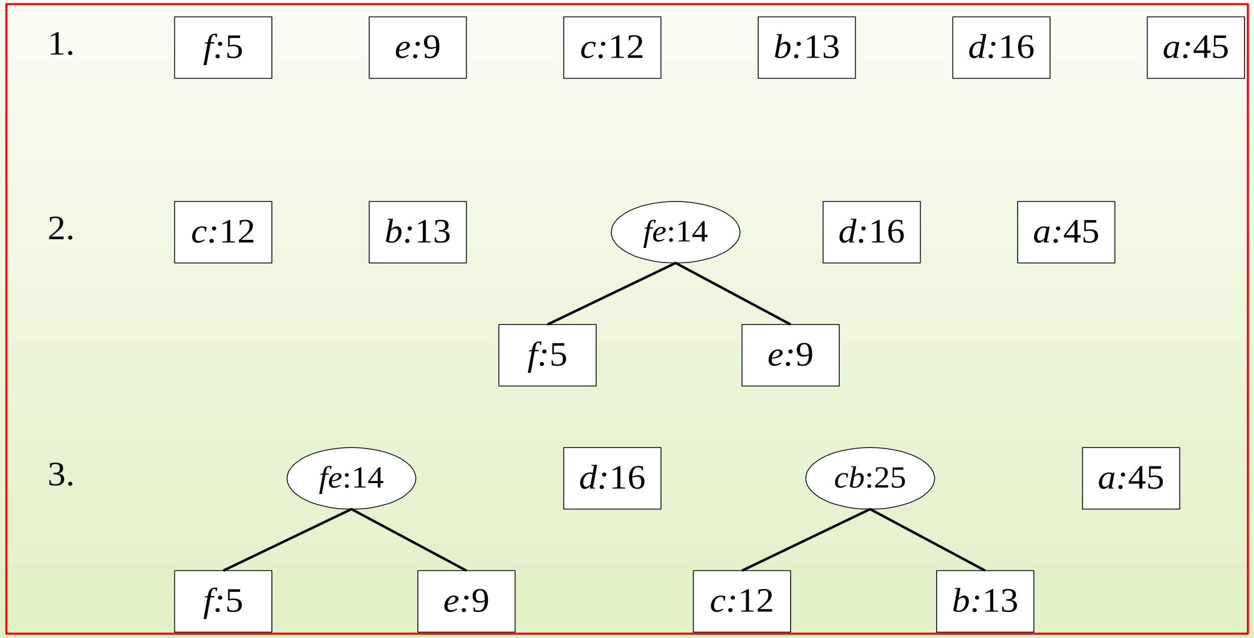
1. **Hitung frektuensi kemunculan semua karakter dalam data nyatakan sebagai pohon bersimpul tunggal.**
2. **Terapkan strategi *greedy* :**
   - **gabungkan dua buah pohon yang mempunyai frekuensi terkecil pada sebuah akar.**
   - **Akar mempunyai frekuensi jumlah dari frekuensi dua buah pohon penyusunnya.**
3. **Ulangi langkah 2 sampai hanya tersisa satu buah pohon Huffman.**

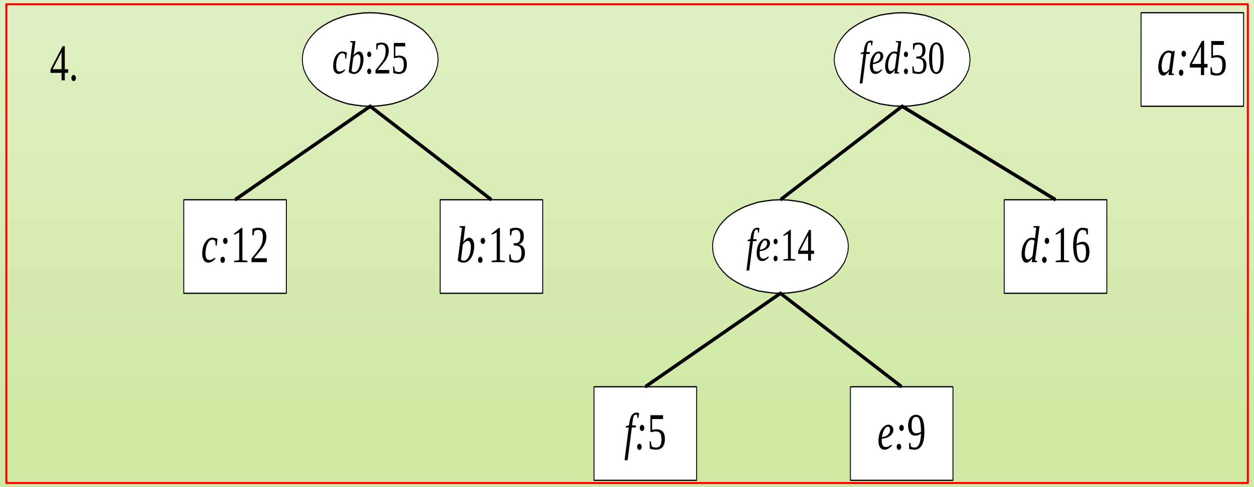
Kompleksitas algoritma Huffman: O(n log n) untuk n karakter.

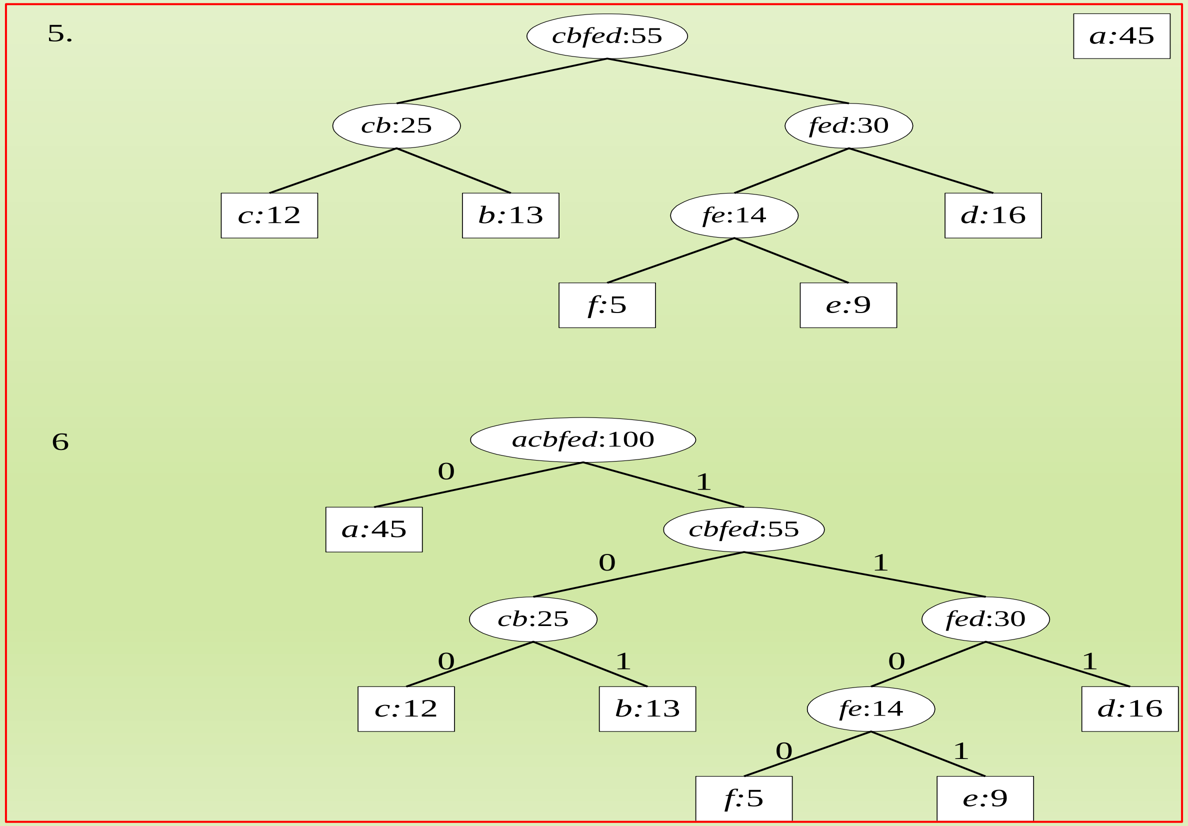
Algoritma Huffman

Contoh

Karakter	***a***	***b***	***c***	***d***	***e***	***f***
Frekuensi	**45**	**13**	**12**	**16**	**9**	**5**







Latihan dan diskusi

Diksukusikan dan kerjakan LKK4 masing-masing kelompok dan kumpulan jawaban di akhir kuliah.

FILKOM
FAKULTAS ILMU KOMPUTER

Selamat Bejalar